An Open Middleware for Proactive QoS-Aware Service Composition in a Multi-Tenant SaaS Environment

Kristof Geebelen*[1], Stefan Walraven[1], Eddy Truyen[1], Sam Michiels[1], Hendrik Moens[2], Filip De Turck[2], Bart Dhoedt[2] and Wouter Joosen[1]
[1]IBBT-DistriNet, Dept. Computer Science, KU Leuven, Belgium
[2]IBBT Dept. of Information Technology, Ghent University, Belgium

Abstract— *Business Process as a Service (BPaaS) is expected to emerge as the next major category of Cloud Computing. The combination of fast expanding technologies with increasing competitiveness will make more and more companies move parts of their business processes into the cloud. This enables them to take advantage of the cloud provider's expertise and to reduce their cost by sharing resources to exploit economies of scale. In such a multi-tenant setting, the cloud provider has to tailor its business processes to meet the QoS requirements of each tenant. This requires techniques for QoS-aware service selection and prediction. However, there is no one-size-fits-all approach to tackle this challenge. The choice of the most suitable technique typically depends on the particular context and characteristics of the services that are being composed. This paper has the following contributions: To support the provider with the creation of custom-tailored processes, we propose an open middleware that provides the mechanisms to perform requirement-driven customizations on shared process templates. This open middleware supports easy integration of new and existing service selection and prediction techniques.*

Keywords: Software as a Service; Multi-tenancy; QoS-aware service composition; WS-BPEL.

1. Introduction

Context. With the trend of cloud computing, Software as a Service (SaaS) has become a common delivery model for business applications. A special form of SaaS is Business Process as a Service (BPaaS) [20], where multiple client organisations are served with a single engine hosting multiple business processes [2]. The business model of a BPaaS company is to offer highly customizable business processes to client organisations (also known as tenants). However, tenants will preferably outsource their business processes to the one provider that can not only offer the highest flexibility in custom-tailored business processes, but more importantly, also offers the most competitive service-level agreements.

The underlying service compositions that enact business processes are typically compositions of third-party services that are possibly delivered as cloud-based SaaS applications. In such constellation of third-party service compositions, guaranteeing SLAs is a difficult problem due to the lack of certainty on the actual QoS values offered by the participating services and underlying cloud platforms. To assist the BPaaS provider with this problem, techniques for QoS-aware service selection and prediction of SLA violations constitute an interesting direction of research.

Problem Statement. Different classes of service selection and prediction techniques have been developed in the past. Existing studies show that their accuracy is still suboptimal and highly variable depending on a.o. the type of composition and the services to be composed ([5], [12], [18]). It can therefore be expected that the choice of the most suitable technique depends on the particular context and characteristics of the services that are being composed. However, no practical service composition framework exists that allows flexible integration with different prediction and QoS selection algorithms. The existing practical frameworks are all based on a specific particular prediction and/or selection algorithms. Moreover, they are closed implementations that cannot be easily modified with new algorithms.

Approach & Contribution. In this paper we present the BPELOnRails middleware for on-demand and run-time generation of customizable BPEL processes in which it is easy to switch between different prediction and service selection algorithms, even at run-time. The key element of our architecture is to adopt the model-view-controller (MVC) pattern [14]. The decision of which services to select can be entirely encapsulated in a separate controller component that can be implemented by means of a general-purpose programming language.

This paper has the following contributions: (a) we present a middleware that allows easy integration of new and existing service selection and prediction algorithms to find an assignment of services to workflow tasks according to a tenant's requested functional and non-functional requirements and (b) we show how these algorithms can be practically integrated on top of our middleware.

Structure of the Paper. The remainder of this paper is organized as follows: Section 2 discusses related research and motivates our work. Section 3 lists the key requirements for the generic architecture: portability, deployment-time customization of business processes, run-time adaptation of business processes and an open programming model for service selection and prediction. Section 4 gives an overview of the BPELOnRails middleware architecture and shows how it is used to perform deploy- and run-time business process customization. This section also briefly discusses our prototype implementation on top of OpenESB. Section 5 illustrate our middleware

* Corresponding author: *kristof.geebelen@cs.kuleuven.be*

by means of a multi-tenant BPaaS for workflows in the healthcare domain. Here we zoom in at how the controller component can be programmed to implement QoS-aware customizations. A performance evaluation of the middleware is documented in Section 6. Finally, Section 7 concludes the paper.

2. Related Work & Motivation

2.1 Related Work

In literature, several works can be found that address the challenge of QoS-aware service composition. Surveys are reported in [12], [18]. Two popular fields of research in this domain are QoS-based service selection and QoS prediction.

QoS-based service selection deals with finding an assignment of services to workflow tasks which maximizes a customer related utility function. Typically this boils down to the following optimization problem: given an abstract composite service and a set of candidate services with different QoS values for each task, find a service for each task such that the global composite QoS value satisfies certain Service Level Objectives (SLO). Popular techniques in literature to solve this challenge efficiently are integer programming (Zeng et al., 2004) and genetic algorithms [4]. These works tackle the composition problem assuming fixed QoS attributes for the elementary services. More recent works take into account that in business environments QoS attributes rarely remain unchanged over the lifetime of a web process and focus on the stochastic service composition problem. Wiesemann et al. [21] formulate the service composition problem as a multi-objective stochastic program which simultaneously optimizes QoS parameters which are modeled as decision-dependent random variables. Their model minimizes the average value-at-risk (AVaR) of the workflow duration and costs while imposing constraints on the workflow availability and reliability. Other approaches ([1], [8]) propose a framework for QoS-aware service composition that does not involve calculating a composite QoS, but support services selection based on constraints specified on the level the workflow tasks.

Driven by the fact that QoS attributes, such as response time, can be very volatile with respect to time, the challenge of QoS prediction has recently gained popularity. Its main goal is to bridge the time gap between the service selection process and the actual execution of the composite service. By predicting accurate expected values for quality measures in the near feature, this technique is able to improve the probability that the selected composition of services, will still respect the SLA constraints during the execution of the workflow. Related work is done by Rosario et al. [17]. They propose QoS estimation based on soft contracts. Soft contracts are characterized through probability distributions for QoS parameters. Chen et al. [7] propose a dynamic QoS model to represent the time related characteristics of QoS, which is used for QoS prediction. In previous work [10], we also focused on this challenge by proposing a theoretical framework to predict the violation probability of an SLA, given the composite service and the monitored QoS values of the participating services. Also proactive QoS monitoring and failure prevention techniques can be categorized in this research field. Canfora et al. [4] support runtime replanning if the actual QoS is predicted to deviate from the required SLA. Leitner et al. [15] propose a framework called PREvent for event-based monitoring and prediction of SLA violations and automated runtime prevention by triggering adaptation actions in service compositions. ProAdapt [3] (Proactive adaptation of service composition) is an approach for proactive adaptation of service compositions due to changes in service operation response time, or unavailability of operations or services. The approach uses exponentially weighted moving average (EWMA) to model service operation response time. Decision for replacement takes into account the predicted response time and cost values as well.

2.2 Motivation

As discussed above, many algorithms or techniques are described in literature that address the challenges in QoS-aware service composition. All of them have their specific advantages and disadvantages depending on the context in which they are used. Important criteria for preferring one technique above others are time complexity and accuracy. More complex algorithms are usually slower but provide more accurate results. In the context of QoS-Aware service composition this trade-off depends on the type of composition. For example, for real-time business processes performance is more important than for long-running processes, possibly taking several days or even weeks to complete. For the latter, restarting the process is usually very costly and choosing an algorithm with a good accuracy will quickly pay-off. Also the type of services that are used in the composition can drive the trade-off. This is mainly the case for prediction algorithms. For example, Geebelen [10] uses a kernel-based algorithm that is able to take into account service dependencies and allows the modeling of seasonality in QoS attributes. The evaluation shows that this technique performs well on predicting response time violations of compositions consisting of services that require human interaction. These services have a low response time during their fixed working hours and a high response time outside working hours and during the weekends. This kind of prediction technique is however less worthwhile in case of automatic services with almost random QoS behavior. Another work [5] reports an empirical study aimed at comparing different QoS prediction models on time series of response times collected by monitoring invocations of 10 services for 4 months. Their results show that the accuracy of a prediction technique depends on the service that is used. Next to performance related criteria, also the scope of the technique can be an important selection criteria. For example, some techniques try to optimize deploy-time service composition, while others address run-time issues. There are works that focus on task-level QoS constraints and others try to enforce global QoS constraints.

As indicated above, no single selection or prediction technique is suitable for all cases. As such there is a need for flexible support that enables integration with different selection and prediction techniques. A limitation of the existing work,

reviewed in Section 2.1, is that a lot of these works are purely theoretical results that have not been validated in a practical setting with existing standards-based service orchestration languages such as WS-BPEL. The little other works, which have implemented a practical system for QoS-aware service composition, support one specific selection and/or prediction technique, require a customized execution engine ([3], [22]), and, more importantly, are closed implementations ([3], [1], [8], [22]) that cannot easily be modified or extended with an alternative technique. In a multi-tenant setting, however, it is crucial that a BPaaS provider offers high flexibility with respect to its tenant's requirements. Therefore, there is need for an open middleware that is not only able to offer custom-tailored business processes, but also offers the option to select an appropriate technique for optimizing the service composition process. This middleware should thus be open for extensions to allow an easy integration of new service selection and prediction techniques.

3. Requirements

We define the key requirements for such an open middleware for tenant-specific QoS-aware service composition:

Portability - As our aim is to support a practical middleware that works with standards-based business process execution languages such as WS-BPEL, we prefer an approach that does not require modification of the language or the underlying execution environment. The proposed middleware should be easily portable between existing orchestration engines and allow automatic generation of customized BPEL scripts, tailored to the needs of individual tenants. To satisfy this requirement, it must be possible to compose WS-BPEL scripts from reusable modular fragments. WS-BPEL, however, has not been designed with this modularity in mind. Therefore, the proposed middleware supports a composition approach that combines annotation-driven generation from abstract process templates with aspect-oriented modularization principles [6], [13].

Deployment-time customization - Deployment-time customization enables to generate a BPEL script, per specific tenant request, before the process is deployed. Deployment-time customization is driven by a QoS-aware selection of participating services and, as we assume that actual QoS values vary over time, a prediction of the probability that the SLA will not be violated during process execution.

Run-time adaptation - Runtime implies that the generated business process should be able to check at developer-designated checkpoints whether the process is still executing in accordance with the agreed SLA. If there is a slack and SLA violation is estimated high, it should be able to trigger a regeneration of the remaining process execution to an alternative set of services of which the actual QoS values are better than the currently selected services.

Open programming model for selection and prediction - It must be possible to support an open yet disciplined programming model where it is easy to implement different techniques for service composition generation and run-time adjustment. The BPaaS provider must be able to easily integrate an appropriate selection and prediction algorithms based on a classification of which technique is best-suited for which particular context. The knowledge needed for classifying and mapping context (business process characteristics, etc.) to appropriate techniques is out of the scope of this paper.

4. A Middleware for Tenant-Specific Proactive QoS-aware service composition

In this section we present our middleware that provides a solution for the requirements discussed above. The middleware extends our framework presented in [11] with multi-tenancy support and QoS-aware service composition.

The idea is based on similar evolutions in the domain of web design. The intent of web design is to create a website that presents the content to the end users in the form of web pages. To comply with today's expectations of end-users, there is a growing tendency to use dynamic web pages. In contrast to static pages, where the content and layout is not changed with every request, dynamic pages adapt their content on the fly depending on the user's input. Similar, our middleware needs to deal with the static character of WS-BPEL scripts and support the creation of dynamic business processes where the content is tailored depending on which tenant will execute the process. We map concepts of dynamic web design to web service composition. The proposed solution is based on model-view-controller [14], an architectural pattern frequently used in Web applications to separate the data model with business logic from interface to promote reuse and modularization.

4.1 Building blocks: Overview

An overview of the middleware architecture is illustrated in Figure 1. Based on the tenant's requirements, the middleware tailors a standard WS-BPEL process that is deployable on an existing execution platform. The tenant can execute the process by creating instances. The main artifacts of the middleware to configure tenant-specific customizations are the Model, View and Controller. To perform QoS-aware customizations, the middleware integrates a QoS Monitor, Predictor or Selector. We briefly discuss these building blocks:

View (Master Process) - The view or master process is similar to a regular workflow process and specifies the sequence of tasks that need to be executed. Instead of including all specific implementation details, it is designed as a template. When the concrete implementation of a task depends on certain constraints, then only a general reference to the type of the task is included. Binding a concrete implementation to the task reference is done later by the controller according to the customization logic. An example abstract task is: invoke an airline booking service. A possible concrete implementation, called an aspect, is a WS-BPEL fragment for invoking Brussels Airlines' booking service.

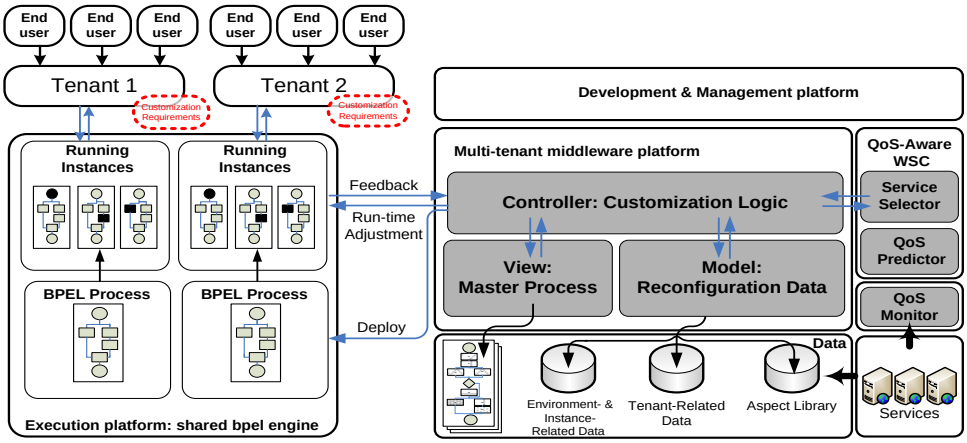

Fig. 1: Overview of the middleware architecture

Model - The model includes an aspect library, tenant-related data and environment- and instance-related data. The library contains aspects for the different WS-BPEL activities that can be modularized as a specific task. All these concrete implementations of a task are bundled in the library and can be reused across workflows. An aspect definition represents a coherent set of basic and structured WS-BPEL activities that realize a particular functionality. Environment-related data specifies properties or parameter values that can be used to evaluate customization logic. An example property is the response time of services participating in a particular workflow. The tenant-related data contains information on the requirements of each tenant which is used to look-up certain constraints when customizations are performed.

Controller - The controller contains the logic to decide which aspects are substituted in the master process. It implements the customization policies that depend on the information available through the model, i.e. the environment-related data and the tenant's requirements. The controller can be implemented using a general purpose language and thus provides the necessary intelligence to implement customizations that require complex algorithms.

QoS Monitor - Quantitative and time-varying QoS attributes for the available concrete services are collected by the monitor to compose time series that can be used by the prediction and selection algorithms. This data is stored in the environment-related database. A detailed discussion of this component is not in the scope of this paper.

QoS Predictor - The Predictor contains the algorithms for QoS prediction. A prediction algorithm uses the monitored data and a tenant's non-functional requirements to predict the violation probability of QoS constraints for a given service composition.

Service Selector - The Selector contains the algorithms for service selection. Such an algorithm uses QoS values (or predictions) to efficiently select suitable services for a composition.

4.2 QoS-aware composition methodology

Our goal is to create a composition, compliant with the requirements of the tenant, using the best available service selection and/or prediction techniques within the given context. Figure 2 shows the different steps of this process. The input, building and output blocks are shown on the left, middle and right respectively. How these blocks are used in our approach is illustrated in between. We distinguish two important phases: the setup, and the actual composition. The latter can be divided in deploy-time customization and run-time adaptation. We briefly go over the important steps:

Setup - All elementary services available to the provider are registered in a repository. (S1) A monitor observes the evolution of the QoS attributes of these services in time and stores them in a database. During the setup phase the *BPaaS provider* has to perform the following actions: (S2) create aspects containing the BPEL syntax to invoke the registered services, (S3) develop a master processes for each composite services he wants to offer to its tenants, and (S4) implement a controller for each master process with a deploy and/or run-time customization strategy using the best suited selection and/or prediction technique for the given context. For example, the provider can implement a combination of selection algorithm A and prediction algorithm B and enforce the results with a deploy-time customization mechanism to maximize the probability that the process will respect a given SLA during its execution.

Deploy-time Customization - (D1) After choosing a composition template (master process) based on abstract services and specifying his QoS requirements, a *tenant* can now trigger the actual composition process. The *middleware* (D2) parses the composition request and (D3) executes the controller logic that belongs to the chosen template in order to tailor a process according to the tenant-specific requirements. If a suitable composition can be created, the resulting executable WS-BPEL process is generated by weaving aspects for concrete service invocations into the abstract composition. (D4) The concrete services are selected by the integrated selection and/or

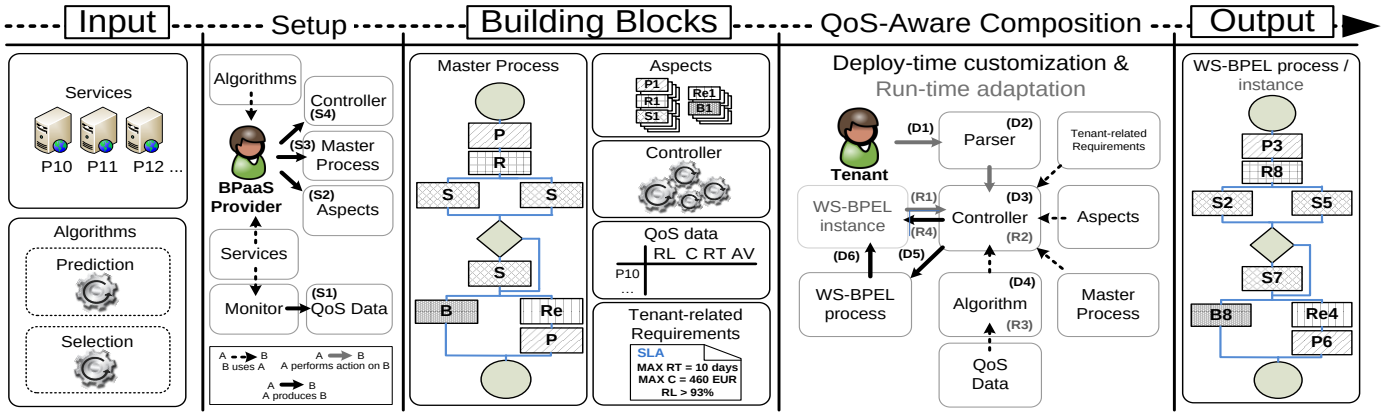

Fig. 2: QoS-aware composition methodology

prediction algorithm. (D5) The resulting WS-BPEL process is deployed on the workflow engine. The tenant will be informed if a composition, with an acceptable chance on fulfilling the agreement, has succeeded. (D6) After receiving the service location, the tenant can now use his tailored composition by creating instances.

Run-time Adaptation - In some scenario's it is beneficial to enforce QoS constraints at run-time. For this purpose, the *middleware* can substitute participating services during execution. This approach can be used separately or in combination with deploy-time customization. Run-time customizations must be annotated in the master process and are also performed by the controller. (R1) On fixed points during execution, the process reports its state to the middleware. (R2) The controller will reevaluate the tenant-specific QoS constraints, given the current state (QoS values) of the already executed tasks. (R3) If there is an indication that the SLA will be violated, a new service selection and/or prediction can be done to find a new combination of services for the remaining tasks that is able to respect the SLA. (R4) In case it is found, the running instance will be adapted and use the new service invocations.

4.3 Prototype implementation

We used standards-based technologies to build a prototype of the middleware. The implementation is done on top of the sun-bpel-engine, a component of the Open Enterprise Service Bus. OpenESB is a Java based open source enterprise service bus. The MVC framework we used for the implementation is the "Ruby On Rails (RoR)" framework [19], known for adding dynamism to web pages. To support different techniques for service selection and prediction, the controller can be implemented using a general purpose language. Our middleware is based on JRuby, a Java programming language implementation of the Ruby language, and thus natively supports Ruby and Java implementations. We also provided a component called "MatlabController" that handles the interfacing with algorithms written in Matlab. A concrete example on how our approach integrates QoS-aware composition techniques is illustrated in the next section.

5. Case Study

5.1 The e-health workflow

Our case study consists of a workflow that realizes a mammography screening program in order to reduce breast cancer mortality by early detection for women above a certain age (see Figure 3). The first task of the workflow consists of sending out invitations to all women that qualify for the program to let a radiologist take images needed for the screening. Once images are taken, the radiologist uploads them to the system (task 2). Next, the image needs to be analyzed by specialized screening centers. There are two independent readings, represented by tasks 3 and 4. These readings can be performed in parallel. In a next step, the two results of the readings are compared. When the results are identical, there is little doubt that the two physicians made the same mistake. Therefore it can be assumed that results are correct and the workflow can proceed with task 5. However, when the results are different, a concluding reading is performed (task 4'). Once the results of the screening of a particular screening subject are formulated, a report is generated (task 5) and different parties are billed (task 6). Finally, a report is sent to the screening subject and her general practitioner in task 7. Task 5 and 7 can be done in parallel with task 6.

The application is hosted by a BPaaS provider, a specialized radiology department. The tenants are other departments of the hospital, other hospitals, or local general practitioners that initiate the breast cancer detection workflow on behalf of their patients, the end-users. Each task in the workflow can be executed by different services, each having their own QoS. Depending on the personal preferences of the tenant or end-user, the requirements for the workflow may differ. The service composition provider can anticipate on these requirements by selecting appropriate services to solve each task. An example customization is where a tenant requires a workflow with a maximal duration of 10 days, a total reliability higher than 93% and a cost of no more than 460 EUR. If such a combination of participating services is possible, our aim is to find and tailor the workflow on such a per-tenant basis by selecting the most appropriate technique.

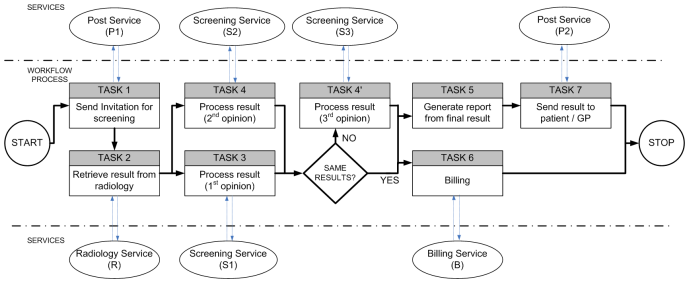

Fig. 3: Example e-health workflow

5.2 Implementing the QoS-aware customization

We show how a BPaaS provider can implement a master process and its associated controller using an appropriate technique (deploy- and/or run-time customization) and algorithm (selection and/or prediction) to realize the scenario discussed above. To illustrate the *openness* of our middleware, we zoom in on how this approach supports easy integration of new and existing service selection and prediction techniques.

In the master process, abstract service invocations need to be specified as annotation-like references to controller methods that implement the customization logic. Two important types of annotations are *"aspect(aspectName, controllerAction)"* and *"aspect_dynamic(aspectName, controllerAction)"*. The former is used for deploy-time customization. Using the latter will insert a callback in the process that contacts the controller during process execution to perform run-time reselection of services.

The controller implements the customization logic using tenant- , environment- and instance-related data. These can be retrieved from the database using the *"get_variable(*)"* method. Returned values depend on the tenant for which the customization is performed. To use service selection or prediction, the controller can call any algorithm that requires input (tenant- , environment-, instance-related data, business process descriptor, etc.) that can be delivered by our middleware and produces output that can be enforced using our deploy- and/or run-time customization mechanisms. Integrating a new algorithm is done by deploying it within the middleware. If it is implemented in a supported language (Ruby or Java), it can be called directly form the controller. Otherwise, our predefined interfacing components can be used (e.g. "MatlabController") or a new one can be implemented.

An example master process and associated controller is shown in Listing 1 and 2. Due to space limitations, we can only illustrate a very simple, but intuitive example. However, it should be clear that the presented constructs offer the flexibility to model much more complex customization scenario's. In this example, the provider opts for using a deploy-time customization technique, where services are selected by using a prediction algorithm that tries to maximize the probability that QoS constraints will be satisfied during the actual execution of the workflow (starting on time *"ST"*). During the tenant-specific customization process, first the *"initialize"* method is executed. This method retrieves the constraints from the database and calls a prediction algorithm in Matlab. The resulting service selection is stored in *"@result"*. Next, the parser executes the methods that weave the aspects in the master process. This is done using *"insert_snippet"* with as parameter the aspect name. The aspects are named in such a way that the last number corresponds with the service numbers that are used by the algorithm. As such, those aspects are selected that contain the invocation syntax for the services that resulted from the prediction.

Listing 1: Master process

```
<process name="e-health">
  <!--# aspect(EHealth, initialize({'processID' => 'e-health'}) #-->
  ...
  <sequence>
    ...
    <!--# aspect(EHealth, insertPost1() #-->
    ...
    <!--# aspect(EHealth, insertBilling() #-->
    ...
  </sequence>
</process>
</aspect>
```

Listing 2: Controller logic

```
@result
def initialize(processID,*)
  ST = get_variable(start_time)
  C = get_variable(constraints)
  @result = MatlabControl.eval("kqoa(processID,ST,C)")
end
def insertPost1(*)
  insert_snippet("invokeP1" + @result[1])
end
...
def insertBilling(*)
  insert_snippet("invokeB" + @result[7])
end
```

6. Evaluation

The goal of this evaluation is to quantify the performance overhead of the different steps is the composition process. Our experimental setup consists of the abstract e-health workflow described in Section 5.1. For each abstract task, there are several candidate services that can execute the task. Each service has its own quality of service characteristics, which vary in time. In this evaluation, we focus on two QoS attributes: response time (RT) and availability (AV), which will be used to evaluate deploy-time customization and run-time adaptation respectively. Services, workflow and middleware were all deployed on an Ubuntu, Intel Core 2 Duo @ 3.16 GHz, 3.2 GiB RAM desktop.

6.1 Deploy-time customization

We consider a scenario where the middleware will try to tailor a service composition at deploy-time, depending on the tenant's required response time (RT) and the planned start time of the execution. The goal is to keep the actual response time of the workflow execution below the required one. When the middleware predicts that no such composition can be created, the composition request is rejected. If the composition request is accepted, the resulting composition is immediately executed by the end-user. The prediction algorithm that is used by the controller is a Kernel-Based Quantile Estimator with Online Adaptation of the Constant Offset (Geebelen et al., 2011).

Performance is measured by decomposing the overhead introduced by each component used in the composition process. The average performance overhead introduced by the middleware to make a deploy-time customization for this scenario was approximately 100 ms. 9% is caused by parsing the tenant's composition request (step (D2) in Figure 2). 9% overhead is caused by evaluation of the customization logic (step (D3)). It takes 12% of the overhead to predict the response time for 1 profile (step (D4)). This assumes that the prediction algorithm is already trained on historical data points. This takes approximately 400 s, but can be done off-line. To create the final composition according to the profile introduces 70% overhead (step (D5)).

6.2 Run-time adaptation

To evaluate run-time adjustments, we consider a similar scenario as in [9]. The radiology task can be completed by three candidate services (R1, R2 and R3). We implemented each of these services so that occasionally a service becomes unavailable for a random amount of time. The frequency and amount of time of unavailability are chosen according to uniform distributions so that the theoretical availability for R1, R2 and R3 matches 75%, 85% and 95% respectively. The middleware selects another radiology center at run-time if the current is unavailable. With this technique, the composition is thus only unavailable after 3 tries, when all three candidates are unavailable. The performance overhead introduced by the middleware to make these run-time customizations was approximately 460 ms. 16% is caused by enabling persistence during workflow execution. Persistence is required by our prototype implementation to enable run-time adaptations. It slows down the execution of the process because the process state is regularly saved to a persistence database. It takes 22% of the overhead to feed back to the middleware to trigger the customization (step (R1) in Figure 2). 2% overhead is caused by evaluation of the customization logic (step (R1&R3)). Each time the process needs to be recovered to use another radiology service (step (R4)), it takes another 275 ms (60% of the overhead).

7. Conclusion

In this paper, we presented an open middleware where a cloud provider can implement context-specific QoS-aware customizations using different techniques and algorithms. The architecture is based on the model-view-controller principle. Based on tenant-, environment-, and instance-related data (Model), the middleware tailors a process from a shared composition template. The shared composition template, called a master process, is designed as a regular WS-BPEL process where tasks can be specified on an abstract level (View). Concrete implementations, modeled as aspects, are selected according to the customization logic using the best suited service selection and/or prediction algorithm (Controller). The middleware, implemented using standards-based technologies, (1) supports *deploy*- and *run-time customization* of WS-BPEL processes, (2) offers *flexibility* to integrate QoS-aware selection and prediction techniques as the enforcement logic is implemented in a general purpose language, and (3) is easily *portable* to different execution environments. To evaluate our middleware, we applied our approach on a case-study in the health-care domain. Our measures confirmed that both deploy- and run-time QoS-aware composition introduce an acceptable performance overhead.

References

[1] Agarwal, V. & Jalote, P. (2010). From Specification to Adaptation: An Integrated QoS-driven Approach for Dynamic Adaptation of Web Service Compositions, In: IEEE International Conference on Web Services, 2010, pp. 275-282

[2] Anstett, T., Leymann, F., Mietzner, R., Strauch, S. (2009). Towards BPEL in the Cloud: Exploiting Different Delivery Models for the Execution of Business Processes. In: IEEE Congress on Services - I.

[3] Aschoff, R. & Zisman, A. Kappel, G.; Maamar, Z. & Motahari-Nezhad, H. (2011). QoS-Driven Proactive Adaptation of Service Composition, In: Service-Oriented Computing, Springer Berlin/Heidelberg, 7084, pp. 421-435

[4] Canfora, G.; Di Penta, M.; Esposito, R. & Villani, M. L. (2005). An approach for QoS-aware service composition based on genetic algorithms. In: Proc. of the Conference on Genetic and evolutionary computation, ACM, pp. 1069-1075

[5] Cavallo, B., Di Penta, M. & Canfora, G. (2010). An empirical comparison of methods to support QoS-aware service selection. In: Proc. of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems, pp. 64-70

[6] Charfi, A. & Mezini, M. (2004). Aspect-Oriented Web Service Composition with AO4BPEL, In: European Conference on Web Services, pp. 168-182

[7] Chen, L.; Yang, J. & Zhang, L. Kappel, G.; Maamar, Z. & Motahari-Nezhad, H. (2011). Time Based QoS Modeling and Prediction for Web Services, In: Service-Oriented Computing Springer Berlin / Heidelberg, 7084, pp. 532-540

[8] Christos, K.; Vassilakis, C.; Rouvas, E. & Georgiadis, P. (2009). QoS-Driven Adaptation of BPEL Scenario Execution, In: IEEE International Conference on Web Services, 2009, pp. 271-278

[9] Erradi, A., Maheshwari, P. and Tosic, V. (2006). Policy-Driven Middleware for Self-Adaptive Web Services Composition. In: Middleware 2006, Springer, LNCS, Vol. 4290, pp. 62-80.

[10] Geebelen, D., Geebelen, K., Truyen, E., Michiels, S., Suykens, J.A.K., Vandewalle, J., Joosen, W. (2011). QoS Prediction for Web Service Compositions Using Kernel-Based Quantile Estimation with Online Adaptation of the Constant Offset. Working paper at `ftp://ftp.esat.kuleuven.be/sista/dgeebele/www2011.pdf`.

[11] Geebelen, K., Kulikowski, E., Truyen E., and Joosen W. (2010). A MVC Framework for Policy-Based Adaptation of Workow Processes: A Case Study on Confidentiality. In: IEEE International Conference on Web Services, pp.401-408.

[12] Xianglan H., Yangguang L., Bin X. and Gang Z. (2011). A Survey on QoS-Aware Dynamic Web Service Selection. In: Proc. of 7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), pp.1-5

[13] Karastoyanova, D. & Leymann, F. (2009). BPEL'n'Aspects: Adapting Service Orchestration Logic, In: IEEE International Conference on Web Services, pp. 222-229

[14] Krasner, G.E., and Pope, S.T. (1988). A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System, tech. report, ParcPlace Systems, Mountain View, Calif.

[15] Leitner, P.; Michlmayr, A.; Rosenberg, F. & Dustdar, S. (2010). Monitoring, Prediction and Prevention of SLA Violations in Composite Services. In: IEEE International Conference on Web Services, pp. 369-376

[16] Mell, P., Grance, T. (2011). The NIST Definition of Cloud Computing. National Institute of Standards and Technology (NIST). Special Publication 800-145 (Draft).

[17] Rosario, S., Benveniste, A., Haar, S., Jard, C. (2008). Probabilistic QoS and Soft Contracts for Transaction based Web Services. IEEE Transactions on Services Computing 1 (4), pp. 187-200.

[18] Strunk, A. (2010). QoS-Aware Service Composition: A Survey. In: IEEE 8th European Conference on Web Services (ECOWS), pp. 67-74

[19] Thomas, D.,Hansson, D.,Breedt, L. and Clark, M. (2007). Agile Web Development with Rails, 2nd Edition.

[20] Wang, M., Bandara, K. Y., and Pahl, C. (2010). Process as a Service - Distributed Multi-tenant Policy-based Runtime Governance. In: IEEE International Conference on Services Computing. IEEE Computer Society, pp. 578-585.

[21] Wiesemann, W., Hochreiter, R., Kuhn, D. (2008). A Stochastic Programming Approach for QoS-Aware Service Composition, In: Int. Symposium on Cluster Computing and the Grid, CCGrid 2008, Lyon, France, pp. 226-233.

[22] Zeng, L., Benatallah, B., Ngu, A. H. H., Dumas, M., Kalagnanam, J., and Chang, H. (2004). QoS-aware middleware for web services composition. In: IEEE Transactions on Software Engineering, vol. 30, no. 5, pp. 311-327.